PROJECT: PARAMETRIC MODELING WITH OPENSCAD

LEARN TO CODE FOR 3D PRINTING: MAKE A NAMETAG

BACKGROUND

In this project, students will learn and apply basic programming skills with the OpenSCAD language to style and customize a nametag. Visually, OpenSCAD is different from other types of 3D modeling programs. It's a simple declarative computer language that was built specifically for designing 3D printable models. By modifying existing OpenSCAD code for a wavy nametag, students will explore parameters, dimensions, "for" loops, translations, and boolean operations.

This project is suitable for all grades and ages, with no previous coding experience required. It can also be scaled up for more advanced students to explore the principles of geometry, trigonometry, and calculus.

SCOPE

Students will learn how to write basic code to generate objects with the OpenSCAD language. Once comfortable, they'll explore a sample coding project that prompts them to discover how changing different variables affects their 3D model. Finally, they'll add their own unique variables and functions to further customize a nametag.

PROJECT OUTLINE

Investigate: Parametric and Customizable Models
Explore: Modeling with OpenSCAD Code
Create: Customize a Nametag Using OpenSCAD Code
Create: Write OpenSCAD Code from Scratch to Design a Model
Further Activities: Customizer, Trigonometry, and Calculus

LOGISTICS

- Technology
  - MakerBot Replicator
- Suggested print time: 20–35 minutes per nametag
  - Computers with OpenSCAD and MakerBot Desktop installed
- Download OpenSCAD at www.openscad.org. This project requires OpenSCAD version 2015.03 or higher.
- Fully commented code for this lesson is in the file **MakerBot_NametagCode_Project.scad**, downloadable from the MakerBot Learning account on Thingiverse.

- For an easier variant of this project, or as a backup in case of local technology issues, the nametag design can be accessed from the MakerBot Learning account on Thingiverse. By selecting **Open in Customizer** from this link, the students can design their own nametags without having to interact with the code itself.

LEARNING OBJECTIVES

General

- Confidence writing basic code with simple parameters
- Understanding measurement and dimensions

3D Design (Parametric Modeling)

- Modifying parameters
- Basic OpenSCAD code
- Translation

TERMINOLOGY

- **Variable:** Symbol that signifies a value that can be fixed or changed depending on its definition
- **Function:** A body of code that returns a value or action
- **Parameter:** A variable within a set boundary
- **Facets:** Flat surfaces that make up the outside of an object. The more facets, the smoother the object.
- **Debug:** To search for and fix incorrect portions of code
- **Customizer:** Program built into Thingiverse.com that allows OpenSCAD files to be uploaded as user-editable models
- **Render:** To generate an output based on code written. In OpenSCAD, the render output is a solid object.

INVESTIGATE: PARAMETRIC AND CUSTOMIZABLE MODELS

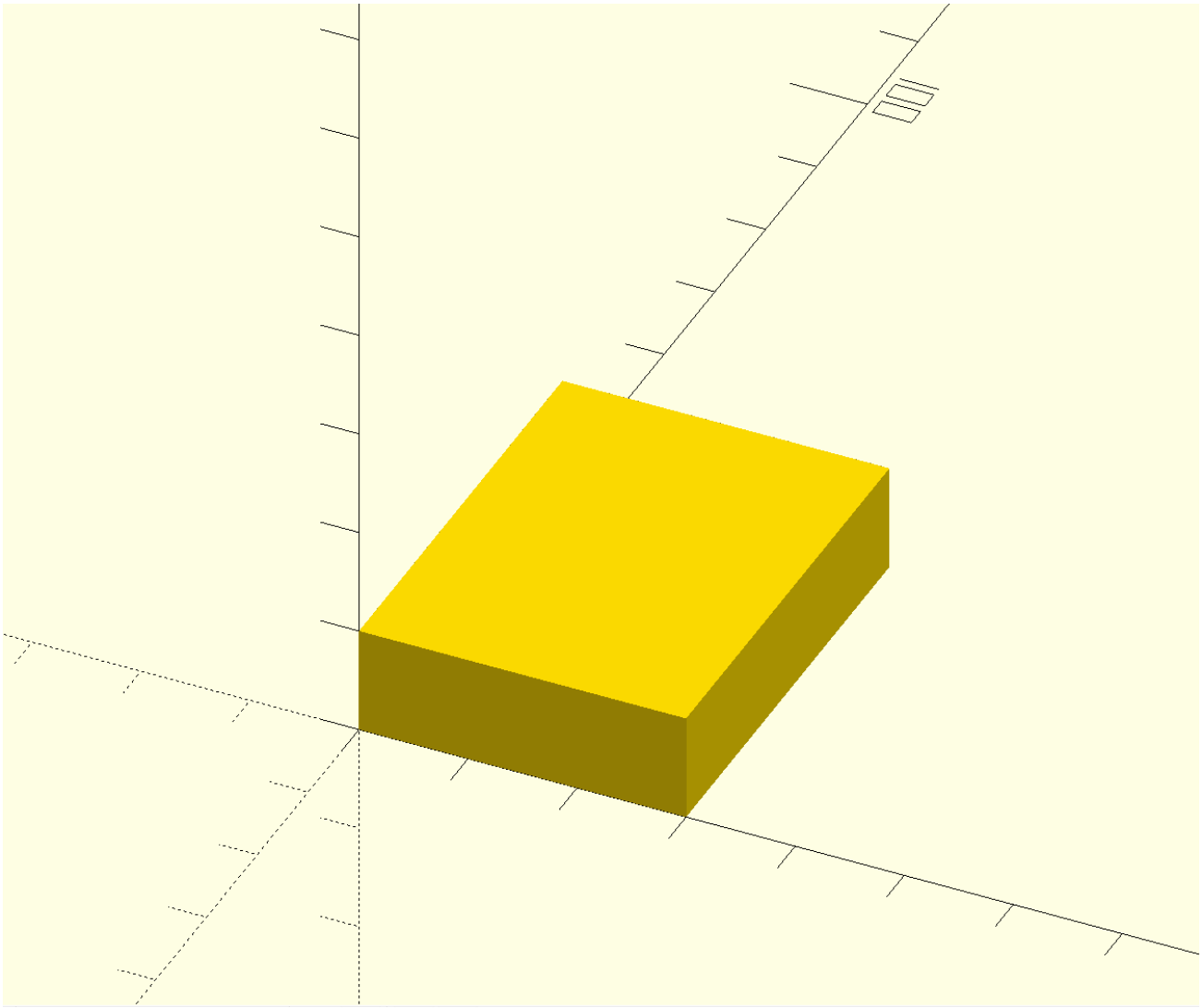
One of the advantages of designing models with code is that you can make them **parametric**. This means you can have elements in your model that are easy to change, like the space between two parts or the length of a lever. The design process is all about iteration: you design something, try to print it in real life, learn from your model, redesign your model, and repeat. Parametric models help shorten the design process.

Before you get started with your own code, take a look at some of the customizable models on **Thingiverse**. Each customizable model was designed with OpenSCAD code that was then put into MakerBot Customizer, which provides an easy user interface for modifying parameters in the model.

- Go to **www.thingiverse.com**. Select the **Explore** tab and then choose **Customizable Things**. You'll see a wide variety of models that Thingiverse users have contributed.
- If desired, change the second category from **All** to another one, for example **Toys and Games**.
- Now choose something for the third category, for example, **Dice**. Many people have contributed models to **Thingiverse** for customizable dice! Choose one of the models, and once the model is open, press the button that says **Open in Customizer**.
- Within **Customizer**, you'll be able to modify whatever parameters the designer has made parametric, or changeable, in their model. Each model will have different parameters that will vary based on what the designer has enabled. Play around with **Customizer**; every time you choose or change a value on the left interface, the picture on the right will update to reflect your changes.
- You can also look at the code that the designer used to make their model by clicking on the **View Source** button that's under the picture of the model in **Customizer**. Some models are made with very complicated code, and some are surprisingly simple.
- Investigate two or three other customizable models of different types, changing parameters and peeking at the code for each one. There is great power, flexibility, and variety in designing with code. What would you design with this type of tool?

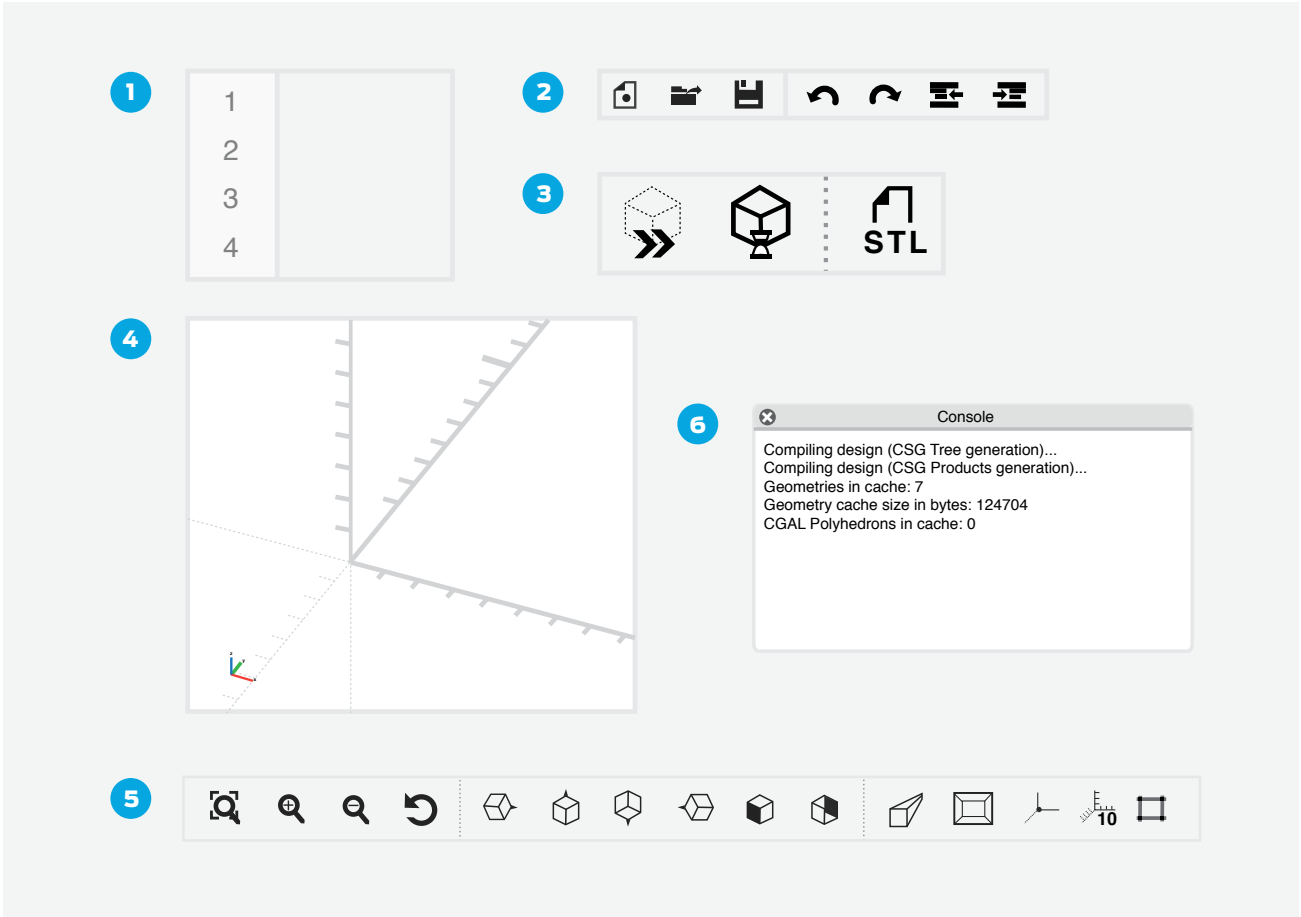
EXPLORE: MODELING WITH OPENSCAD CODE

When you open the OpenSCAD software, you'll see a window like the one shown. The basic workflow is to type code into the **editor window** on the left, press **F5 on a PC (or Function-F5 on a Mac)** to compile the code, and then look in the **view area window** on the right to see the result. A small **console log** at the bottom right will display output notes and sometimes error messages. You can use the mouse to navigate in the **view area**. Follow the steps in this section to get comfortable with how OpenSCAD works.



INTERFACE:

1. **Editor window** – Type code here to define your model
2. **System options** – Save your file, undo/redo actions, and format your code
3. **Model options** – **Preview**, **Render**, and **Export STL** files from this menu
4. **View area** – Displays the model defined in the editor window
5. **View menu** – Change your viewing angle using your mouse or the buttons in this menu
6. **Console** – Displays output notes and error messages



STEP 1: TYPE AND COMPILE YOUR FIRST OPENSCAD COMMAND

- Type **cube([30,40,10]);** in the editor window. Be sure to type it exactly as written.
- Press **F5** (or **Function-F5**) to see the resulting cuboid. As an alternative to using F5, you can use the **Preview** button in the **Model Options** menu.
- If your axes and/or scale markers are not visible, turn them on from the **View** menu.
- Notice that the cuboid is **30 mm** in the **x-direction**, **40 mm in the y-direction**, and **10 mm** the **z-direction**.

STEP 2: NAVIGATE AROUND THE OBJECT IN THE VIEW AREA.

- **Orbit** – *Left-click and drag* to orbit around the model.
- **Pan** – *Right-click and drag* to move the view around.
- **Zoom** – *Scroll the mouse wheel* up and down.

STEP 3: GET COMFORTABLE WITH BASIC OPENSCAD PRIMITIVES.

- Try typing the **primitives** commands in the table below into your OpenSCAD editor window and then press **F5** (or **Function-F5**) to see what happens.
- Experiment with modifying the code to get a feel for how the commands work.

CODE EXAMPLE	WHAT THE CODE DOES	NOTES
cube(30);	Creates a cube centered at the origin with all sides of length 30mm.	The semicolon tells OpenSCAD to draw the object. You'll need one at the end of many of your lines of code.
cube([30,40,10]);	Creates a cuboid with one of its corners at the origin and side lengths of 30 mm, 40 mm, and 10 mm.	The square brackets denote [x,y,z] coordinates. The round brackets (parentheses) pass information to the cube command.
cube([30,40,10], center=true);	Creates the same cuboid as above but with the center of the cube at the origin.	Sometimes one type of centering is more convenient than another.
sphere(20, $fn=24);	Creates a sphere with radius 20mm, centered at the origin, and with 24 facets around the equator.	Increasing the number of facets makes a smoother sphere, but can make your code take longer to compile.
cylinder(h=20, r=10, $fn=40);	Creates a cylinder with height 20mm and radius 10mm, centered at the origin, and with 40 facets around the equator.	Try setting $fn=6 or $fn=4; with a low number of facets so your cylinder can be a hexagonal or square prism.
cylinder(h=20, r1=10, r2=5);	Creates a truncated cone with height 20mm, lower radius 10mm, and upper radius 5mm, centered at the origin.	Note that when you don't specify $fn, OpenSCAD uses a default value.

STEP 4: GET COMFORTABLE WITH BASIC OPENSCAD MODIFIERS.

OpenSCAD also has commands for moving, scaling, extruding, and combining primitives. These are powerful tools for turning primitives into complete designs.

- Try typing the code snippets below into the **editor window**, and press **F5** (or **Function-F5**) to see what happens.
- Experiment with modifying the code to get a feel for what the commands can do.

CODE EXAMPLE	WHAT THE CODE DOES	NOTES
translate([-20,0,30]) cube(20, center=true);	Creates a 20mm cube centered at the origin, and then shifts the location of that cube –20 mm in the x-direction and 30 mm in the z-direction	The translate command modifies whatever primitive immediately follows it in the code. Note that translate itself doesn't get drawn, so it has no semicolon.
rotate(45, [0,1,0]) cube(20, center=true);	Creates a 20mm cube centered at the origin, and then rotates that cube 45 degrees around the y-axis (along the vector [0,1,0])	You may want to left-drag in the view area to look around the object and see how it was rotated.
linear_extrude(h=20) circle(20, $fn=6);	Creates a "circle" of radius 20 mm with only six facets (in other words, a hexagon), and then extrudes that shape upward for 20 mm.	Notice the use of the 2D primitive circle. The command linear_extrude only works on 2D objects.
linear_extrude(h=20, twist=60) circle(20, $fn=6);	Creates the same "hexagon-circle" with radius 20mm, and then extrudes that shape upwards for 20mm while rotating a total of 60 degrees.	The linear_extrude command is a modifier, not a primitive, so it does not need a semicolon.

difference(){
  cube(20, center=true);
  translate([0,0,-20])
  cylinder(h=50, r=8);
}
	Create a 20 mm cube and then remove a cylinder shape from it. The cylinder is translated downward so it penetrates through the cube.	The difference command draws the first object and then subtracts every other listed object from the first.

STEP 5: MAKE SIMPLE OBJECTS WITH OPENSCAD CODE.

Now use the code above to make some basic objects. See how many of the following objects you can make with OpenSCAD code. Remember to press **F5** (or **Function-F5**) after each update to your code so you can see the result.

- A 30 mm sphere that's shifted –10 mm in the x-direction, 40 mm in the y-direction, and 15 mm in the z-direction.
- A cuboid that measures 10 mm in the x-direction, 100 mm in the y-direction, and 5 mm in the z-direction.
- Two cylinders that intersect at right angles to each other.
- A pointy cone shape with another pointy cone shape underneath it in the opposite direction (like two ice-cream cones stuck together around their rims).
- A box with holes in each of its sides. You can use three cylinders to form the holes. Can you align the holes exactly to the center?
- An octagonal prism that twists 30 degrees from its base to its top.
- An open box with three small spheres inside it.
- A snowman!

STEP 6: EXPLORE MORE WITH ONLINE OPENSCAD TUTORIALS AND DOCUMENTATION.

- David Dobervich's *OpenSCAD Tutorial #1* video on YouTube
- Patrick Conner's *Welcome to OpenSCAD* video on YouTube
- OpenSCAD documentation, Cheat Sheet, and list of Getting Started Tutorials http://www.openscad.org/documentation.html

CREATE: CUSTOMIZE A NAME TAG USING OPENSCAD CODE

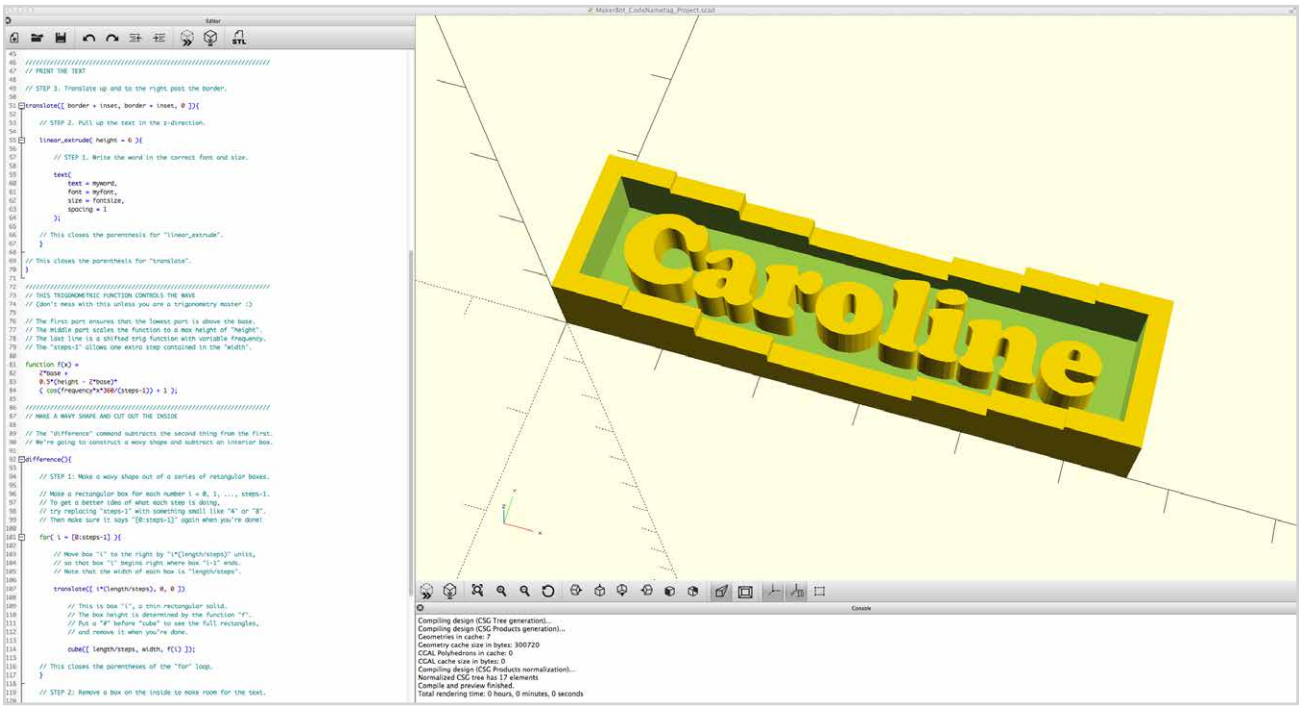
In this project, you'll design and 3D print a wavy-bordered nametag by modifying parameters in OpenSCAD code. You'll also dive into the code and learn how these parameters create the nametag design. Advanced students may want to change the code itself to create even more varied designs.

STEP 1: TRY TO FIGURE OUT WHAT THE CODE IS DOING.

Fully commented code for this lesson is downloadable from the MakerBot Learning account on Thingiverse.

Open the file **MakerBot_NametagCode_Project.scad** from MakerBot Learning on Thingiverse and look through the code in the **editor** window.

Don't type or change anything yet; just scroll through, look at the text, and try to figure out what might be going on in the code. The code should look something like the image below.



Discussion

- In groups, discuss the code among yourselves, sharing ideas about what each part of the code might be for.
- Each group should write down at least three ideas to share with the class.
- Discuss each group's ideas as a class, trying to decode the code together.

STEP 2: MODIFY TEXT AND FONT PARAMETERS

Near the top of the code are three sets of parameters that students can use to customize their nametag. The first set of parameters concerns the text, font, and font size of the nametag:

```
// TEXT AND FONT
myword = “Caroline”;
myfont=”Phosphate:style=Inline”;
fontsize = 8;
```

You can choose to use your name or word in **myword**, but keep in mind that longer names and words will take longer to 3D print. The **myfont** parameter uses the font names of the fonts installed on your local system. Use the **Help / Font List** pull-down menu to obtain a list of your available fonts. You can drag and drop font names directly from this list into your OpenSCAD code or copy/paste into the code. You should not change **fontsize** unless the student chooses a particularly large or small font shape.

Remember that after each change, you must press **F5** (or **Function-F5**) or the **preview icon** to see the updated result in the **view area**. Compiling errors at this stage are usually due to lost quotation marks, missing semicolons, and incorrectly spelled font names.

STEP 3: ADJUST DIMENSIONS TO MATCH THE TEXT

The next set of parameters sets the overall size of the nametag:

```
// TOTAL DIMENSIONS
length = 56;
width = 15;
height = 8;
```

The **length** and **width** parameters will have to be adjusted depending on the size and length of your word choices in the previous step. The **height** parameter determines the height of the tallest part of the border wave. Compiling errors at this stage are usually due to lost semicolons.

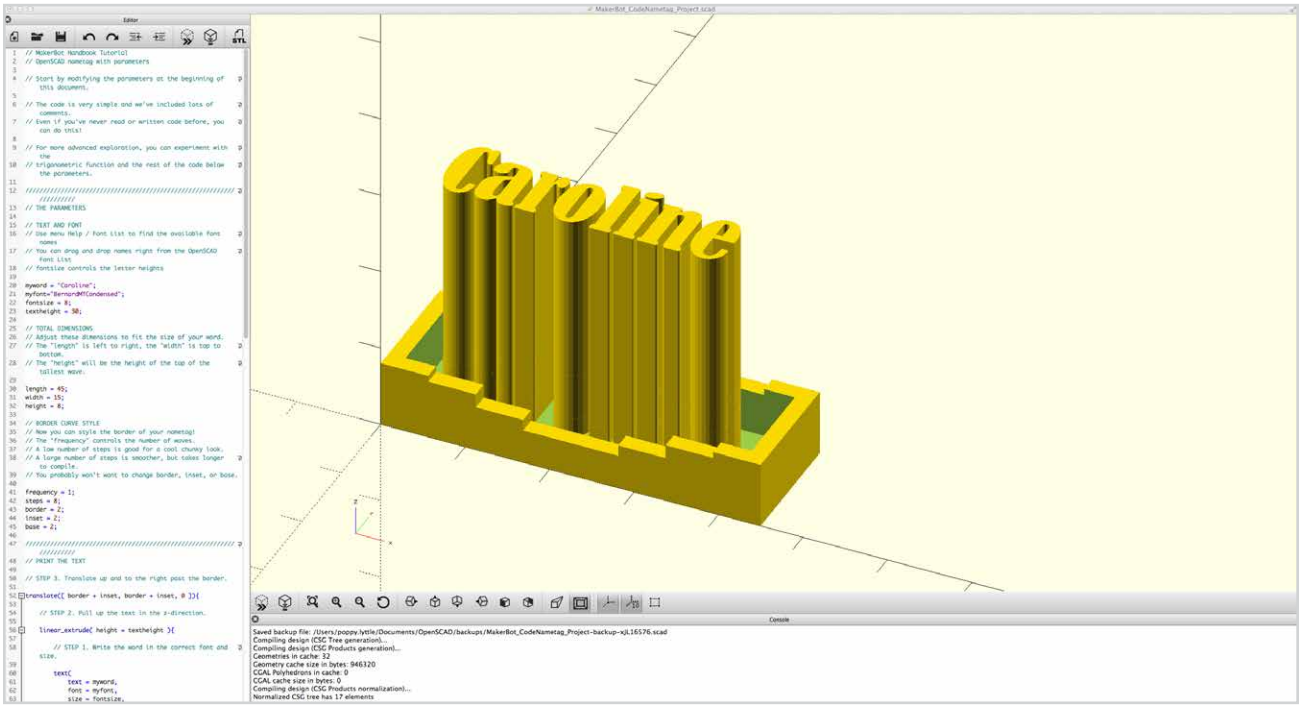
STEP 4: STYLE THE WAVE OF THE NAMETAG BORDER

The final set of parameters determines the style of the wavy border around the outside of the nametag:

```
// BORDER CURVE STYLE
frequency = 1;
steps = 8;
border = 2;
inset = 2;
base = 2;
```

The wavy border's shape is determined by a trigonometric function, but you do not need to know any trigonometry to modify these parameters. The **frequency** parameter changes the number of waves. The **steps** parameter controls the number of times the wave height changes; lower values make stair-step shapes and higher values make smoother curves. You'll likely not need to change the **border**, **inset**, or **base** parameters, but feel free to explore adjusting them to see what happens.

STEP 5: CREATE A NEW PARAMETER FOR TEXT HEIGHT



So far, you've just been modifying existing parameters in the OpenSCAD code. In this step, you'll create a new modifiable parameter. Look at the second section of the code, below the //PRINT THE TEXT comment on line 47. Notice that the code is read in reverse order: First the 2D text is created, then the letters are extruded upward, then the extruded letters are shifted up and to the right so they do not intersect the border.

- Determine the part of the code that controls the height of the letters, and figure out what to change to make the letters shorter or taller. (Hint: The text height starts out set to 6 mm.)
- Replace the numerical value on line 55 with a named parameter called **textheight**.
- Add a parameter definition to the top of the document on line 23 by typing **textheight = 6;**
- Now you can modify the text height by changing the value of this parameter on line 23.

STEP 6: CREATE A NEW PARAMETER FOR CHARACTER SPACING.

Now repeat the same process to add a new parameter called **myspacing** that controls the amount of space between the characters of the printed name.

- Try to find the place where a **myspacing** parameter would go in the code. (Hint: it's part of the **text** command and it starts out set to 1.)
- Once this parameter is set up, modify its value to adjust the character spacing to your liking. The default value of 1 represents normal spacing. Note that even small changes like modifying to 1.2 or .9 can change the spacing considerably.
- You may have to readjust the **width** of the nametag if the student makes substantial changes to the **myspacing** parameter.

STEP 7: OUTPUT AN STL MESH FOR 3D PRINTING.

Once you're done designing your nametag, export the designs to **STL** files for 3D printing:

- First, to be safe, save your OpenSCAD code file and/or write down notes about the values of the parameters you chose in your design. The exported **STL** file will not save or record those numerical values, and if there's a problem with an **STL** file, it's good to have a way to reconstruct the basic design again.
- To generate a 3D mesh suitable for printing, press **F6** (or **Function-F6**) or use the **Render** button. This process can take a lot longer than **Preview**, especially if you've chosen high values for the **step** parameter. Watch the progress bar at the bottom right of the screen.
  *Note: The **F5** action only generates a preview of the 3D object in the view area, not a complete render.*
- When the **F6** render has completed, use the pull-down menu or the **Export STL** button to export the model as an **STL** file.
- The nametag **STL** files can now be imported into **MakerBot Desktop** and 3D printed, either individually or with multiple nametags on one build plate.

CREATE: WRITE OPENSCAD CODE FROM SCRATCH TO DESIGN A MODEL

In this project, you'll write your own OpenSCAD code from scratch, making use of parameters. You can make anything you want, but it's a good idea to start with a simple design such as creating a snowman or a box and lid.

STEP 1: PLAN, THINK, AND CODE

- Before you start writing code, think about what you want to make. What parameters do you want to be able to modify in your design?
- While coding, watch out for missing semicolons, incorrect capitalization, misspellings, and paren theses that didn't get closed correctly.
- Save your work and compile with **F5** (or **Function-F5**) often. It's easier to **debug** code (fix problem sections) that has only one new line in it than it is to debug whole blocks of new code. If you compile more often, then you'll be able to target your bugs much faster.
- When approaching OpenSCAD code, it's a good idea to start by explicitly defining your parameters (**spacing = 1**). Then, if desired, go back and create variables for specific parameters that can be easily changed across your project (**spacing = myspacing; myspacing = 4**).

STEP 2: TRY NEW THINGS

- OpenSCAD has a powerful polygon() function that you might find useful in your design. For more information, see the OpenSCAD documentation about this function:
  - http://www.openscad.org/documentation.html
- In addition to linear_extrude(), there's a command called rotate_extrude() that you can use to turn 2D shapes like squares, circles, and polygons into 3D rotated objects. It works a little unexpectedly, so be ready to do some experimenting. For more information, see the OpenSCAD user manual.
- Advanced students can experiment with for() loops and if...then statements in their code, or explore how to use a module. For more information, see the OpenSCAD user manual.

STEP 3: EXPORT, PRINT, TEST, REPEAT

- When you're done designing, press **F6** (or **Function-F6**) to render your mesh model, and then **Export STL** for printing.
- Test your model. Does it work the way you want it to? Are there any parameters that you can tweak to make it better, or anything that you want to add? Remember that 3D printing is great for iterating your designs. This means that you might not get a model exactly right on the first try. Every failure is a chance to learn something and make your design better. Be willing to try, fail, and try again until you get the results you want.

FURTHER ACTIVITIES: OPENSCAD AND THINGIVERSE CUSTOMIZER

ACTIVITY 1: MODIFY OPENSCAD CODE FOR THINGIVERSE CUSTOMIZER.

Thingiverse Customizer allows users to modify parametric designs without having to interact with OpenSCAD code directly. It's relatively easy to modify a design so that it can be used in the Customizer interface; the basic idea is to add comments to your OpenSCAD code that Customizer uses to create sliders and input fields for parameters.

Materials

- Computer with OpenSCAD and downloaded code file

Steps

1. Review documentation on making a customizable thing at http://customizer.makerbot.com/docs.
2. Download code file **MakerBot_NametagCode_Project.scad** from the MakerBot Learning account on **Thingiverse**.
3. Compare with the same OpenSCAD code used in the **Thingiverse Customizer** (select **View Source** from the lower right corner of the **Customizer** at the **MakerBot Learning** account).
4. Make modifications as you explore.

ACTIVITY 2: EXPLORE THE TRIGONOMETRY USED IN THE NAMETAG.

Have precalculus students investigate the function used in the wavy border design of the nametags:

```
function f(x) =
  2*base +
  0.5*(height - 2*base)*
  ( cos(frequency*x*360/(steps-1)) + 1 );
```

In particular, what does each of the components of this function do to change the shape of the wave? Consider amplitude, frequency, and transformations. What would happen if the cosine function were replaced with the sine function? What other functions would make good nametag borders?

Materials

- Knowledge of precalculus and basic trigonometric transformations
- Computer with OpenSCAD and downloaded code file

Steps

1. Change and consider what amplitude, frequency, and transformations are doing.
2. Try changing the cosine function to sine. What happens?
3. Explore other functions that could affect the border.

ACTIVITY 3: EXPLORE RIEMANN SUMS.

The wave shape on the border of the nametag design is actually a simple **Riemann sum** approximation of the area under the graph of the function **f(x)** on an interval. Here's the code that produces the steps of the wave.

```
for( i = [0:steps-1] ){
  translate([ i*(length/steps), 0, 0 ])
    cube([ length/steps, width, f(i) ]);
```

Materials

- Knowledge of calculus and the definition of Riemann sums for definite integrals
- Computer with OpenSCAD and downloaded code file

Steps

1. Look at a small number of steps to get an idea of what is happening. What type of Riemann sum is being constructed?
2. Something subtle is happening with the steps-1 part of the function definition. Why is that part of the code here and what does it do for the design? Replace steps-1 with a value (try 4, 8, etc.) and note what happens. What does it mean about the interval being used for the Riemann sum?

KNOWLEDGE CHECKS

- What are the benefits of a parametric design?
- How do you generate basic shapes with code?
- Why is it important to use variables in OpenSCAD code?
- What are examples of parameters you might change when designing an object?
- What do you need to change or add to your OpenSCAD file to allow it to be used in Customizer?

MOVING FORWARD

Picking the right tool for your design process is an important decision. OpenSCAD really shines when you need to create designs with easily modifiable parameters. With OpenSCAD it's easy to go back and change those parameters to fit any situation you may need or make small adjustments to measurements. This flexibility allows you to iterate more on your designs. Exploring OpenSCAD further to expand upon your coding knowledge will give you the freedom to create many complex designs. Then you can take it to the next level by adding your designs to the **Thingiverse Customizer**!